Optimizing Fund Allocation for Game-Based Verifiable Computation Outsourcing

Pinglan Liu, Xiaojuan Ma, and Wensheng Zhang(✉)

Computer Science Department, Iowa State University, Ames, IA 50011, USA
{pinglan,xiaojuan,wzhang}@iastate.edu

Abstract. This paper considers the setting where a cloud server executes tasks submitted by multiple clients. Every client wishes to assure honest execution of the tasks by employing a trusted third party (TTP) to verify with a probability. The cloud server makes a deposit for each task it takes, each client allocates a budget for each task submitted, and every party has its limited fund. We study how to allocate the funds optimally such that: a economically-rational cloud server honestly computes each task; the server's wage is maximized; the delay for task verification is minimized. Game theory is applied to formulate these problems, and optimal solutions are developed. Each solution is evaluated through rigorous proofs. To the best of our knowledge, this is the first work on optimizing fund allocation for verifiable outsourcing of computation in the setting of one server and multiple clients, based on game theory.

Keywords: Outsourcing · Computation verification · Game theory · Optimization

1 Introduction

The popularity of cloud services promotes computation outsourcing. Clients outsource heavy computational tasks (e.g., data mining, machine learning) to a cloud server with rich resources to handle them. The server aims to efficiently utilize its resources and maximizes its profit from the computation. Each client desires to pay no more than a certain predefined budget and gets correct computation results with short latency.

To assure that the server returns correct results, verifiable outsourced computation mechanisms should be in place. A large variety of verification schemes have been proposed in the literature. They can rely on cryptography [1–12], trusted hardware [13–16], redundant system (with at least one trusted server) [17,18], game theory [19–26], or combinations of the above. The approaches purely relying on cryptography or trusted hardware usually have high costs and/or low performance/scalability, while the game-based approaches have been more appealing for their lower costs due to the practical assumption of economically-rational participants. Hence, we also apply game theory in our study.

M. R. Khosravi et al. (Eds.): CloudComp 2021, LNICST 430, pp. 60–71, 2022.
https://doi.org/10.1007/978-3-030-99191-3_6

We adopt the basic model that each client outsources her tasks to only one server (without redundancy) but with probabilistic auditing. Additionally, for each task, the server is required to make a deposit, which can be taken by the client when the server is found misbehaving; each client should prepare a *budget* that includes the wage paid to the server (if the server is not found dishonest) and the cost for hiring a trusted third party (TTP) to check the result returned by the server (i.e., auditing). A relation among the deposit, wage and the auditing probability can be found such that, the server's most beneficial strategy is to act honestly as long as the condition is satisfied.

It is natural to assume the cloud has a certain fund to spend as deposits for the tasks it take. However, the fund is limited at a time and should be spent smartly so that the server can maximize its benefit, which we measure as the wage it can earn. A client is also assumed to have certain fund for the tasks she outsources. The client's fund is limited too, and thus should be smartly spent as well to maximize her benefit, which we measure as the delay that she has to experience when waiting for her tasks to complete. Here, the client's spending strategy includes: first, how to distribute a given amount of fund to the tasks that are submitted simultaneously or within the same time window; second, for each of the tasks, how to further divide the assigned budget for paying the server's wage and for hiring a TTP respectively. *How can we smartly allocate the server and the clients' funds to maximize their profits?* To the best of our knowledge, this is a question that has not been raised or answered in the literature. The focus of this paper is to formulate and solve this problem.

We formulate the problem as follows. First, we formulate a per-task game-based outsourcing model. The model enforces a secure relation among three components, the server's deposit, the server's wage and the client's auditing probability, where the latter two determine the client's budget, to ensure the server's best choice to be compute honestly. In addition, the model has the attractive property that, the wage and the auditing probability are not fixed but functions of the server's deposit and the client's budget; the larger is the deposit and/or the budget, the larger is the wage and the smaller is the auditing probability. Note that, larger wage and smaller auditing probability (and thus shorter delay) are desired by the server and the client, respectively. Second, we formulate the interactions between the server and clients into an infinite extensive game with perfect information. Within this game, the server and the clients are the parties; the different ways to dividing the server's fund into the tasks' deposits and to dividing the clients' funds into the tasks' budgets are the parties' actions; and the parties' utilities are defined as functions of the actions.

To solve the formulated problem, we develop algorithms that find the Nash Equilibria of the games, which are also the optimal solutions that maximize the server's wage and minimize the clients' delays in two settings: there is one client or multiple clients in the system.

In the following, Sect. 2 introduces the system model and the per-task game-based outsourcing model. Section 3 defines the game between the server and the clients. Sections 4 and 5 develop the solutions. Finally, Sect. 6 concludes the paper.

2 System Architecture

System Model. We consider a system consisting of a cloud service provider (called cloud server or *server* hereafter), m *clients* that need to outsource computation tasks to the server, and some trusted third parties (called *TTP*s hereafter) which the clients can resort to for verifying outsourced computation.

The server, denoted as S, is not completely trusted and its execution of the tasks outsourced by the clients may not always be correct. However, we assume the server is economically rational; that is, it always aims to maximize its profit and will not misbehave if that would cause penalty. As to be elaborated in Sect. 2, we introduce a game-based approach to guarantee that the server honestly executes the outsourced tasks. We assume that the server is willing to use a certain amount of fund as deposit to assure its client of its honest behavior.

We denote the m clients as $C_1, \cdots, C_m$. The tasks outsourced by each client C_i are denoted as $t_{i,j}$ for $j = 1, \cdots, n_i$, where n_i is the number of such tasks. Each task $t_{i,j}$ is associated with two costs denoted as $c_{i,j}$ and $\hat{c}_{i,j}$, where $c_{i,j}$ is the server's cost to execute the task and $\hat{c}_{i,j}$ is each TTP's cost to execute the task. To simplify the presentation, we assume the execution time is proportional to the costs; that is, assuming k is a certain constant, the server's execution time of the task is $k \cdot c_{i,j}$ and each TTP's execution time of the task is $k \cdot \hat{c}_{i,j}$. Each client C_i allocates a budget $b_{i,j}$ for each task $t_{i,j}$, where $b_{i,j} \geq c_{i,j}$ so that the server is willing to take the task.

Each TTP can be hired at the price of $\hat{c}_{i,j}$ by a client to check if the server's execution is correct via re-execution. A TTP can also be a cloud server that has a trusted execution environment (TEE) such as Intel SGX enclave.

Finally, we assume that the server, the clients and the TTPs can access a blockchain system so that no any centralized trusted authority is required.

Per-task Game-Based Outsourcing Model. To ensure that the server honestly executes tasks, we adopt a game theoretic approach as follows. For each task $t_{i,j}$, the server should make a deposit of $d_{i,j}$ and client C_i should promise a budget with a certain expected value of $b_{i,j}$.

After the client outsources $t_{i,j}$ to the server, with a probability denoted as $p_{i,j}$ it also hires a TTP to execute the task. After the client has received a result of computation task from the server and/or the TTP, funds are distributed between the client and the server as follows: If no TTP is hired, or the results returned by the server and the hired TTP are the same, the client should pay a wage denoted as $w_{i,j}$, where $w_{i,j} \geq c_{i,j}$, to the server, and the server should also be returned with its deposit $d_{i,j}$. If the results returned by the server and the TTP are different, deposit $d_{i,j}$ should be given to the client. Hence,

$$b_{i,j} = w_{i,j} + p_{i,j} \cdot \hat{c}_{i,j}. \tag{1}$$

Also, as stated in the following theorem (proved in [27]), is the sufficient condition to deter the server from misbehaving and ensure it honestly executes task $t_{i,j}$.

Theorem 1. *If $w_{i,j} \geq c_{i,j}$ and $p_{i,j} \geq \frac{c_{i,j}}{w_{i,j}+d_{i,j}}$, an economically rational server must execute task $t_{i,j}$ honestly and submit a correct result to the client.*

3 Optimization Problem

Game between The Server and The Clients. We model the interactions between the server and the clients as an infinite extensive game with perfect information, denoted as $G = (P, A, U)$. Here, $P = \{S, C_1, \cdots, C_m\}$ is the set of players. A is the set of actions taken by the players, including (i) all possibilities that each C_i can split its budget b_i to n_i tasks and (ii) all possibilities that S can split its deposit fund d to the $n = \sum_{i=1}^{m} n_i$ tasks. Hence, the action set each C_i can take is denoted as $A_{c,i} = \{(b_{i,1}, \cdots, b_{i,n_i}) \mid \sum_{j=1}^{n_i} b_{i,j} = b_i\}$, where b_i is C_i's total budget for its tasks, and each action $(b_{i,1}, \cdots, b_{i,n_i})$ is one possible division of b_i to n_i tasks; the action set S can take is denoted as $A_s = \{(d_{1,1}, \cdots, d_{m,n_m}) | \sum_{i=1}^{m} \sum_{j=1}^{n_i} d_{i,j} = d\}$, where d is the server's fund for deposits, and each action $(d_{1,1}, \cdots, d_{m,n_m})$ is one possible division of d to n tasks. $U = \{U_s, U_{c,1}, \cdots, U_{c,m}\}$ are the players' utility functions.

Constraints on Budgets and Deposit. According to the above definitions of the clients' and the server's actions, the following constraints are obvious:

$$\sum_{j=1}^{n_i} b_{i,j} = b_i, \ \forall\ i \in \{1, \cdots, m\}, \ \textit{and} \ \sum_{i=1}^{m} \sum_{j=1}^{n_i} d_{i,j} = d. \tag{2}$$

For each task $t_{i,j}$, the server's deposit for it should be at least $\hat{c}_{i,j}$, to compensate client C_i's cost for hiring a TTP if the server is found dishonest. Hence, we have the following constraint:

$$d_{i,j} \geq \hat{c}_{i,j}. \tag{3}$$

Regarding budget $b_{i,j}$ for $t_{i,j}$, according to Eq. (1), it includes wage $w_{i,j}$ paid to the server for honest computation and the expected cost to hire TTP. First, based on Theorem 1 and that TTP should be hired as infrequently as possible, we set

$$p_{i,j} = \frac{c_{i,j}}{w_{i,j} + d_{i,j}}. \tag{4}$$

Second, $w_{i,j} \geq c_{i,j}$ must hold to incentive the server. Because $b_{i,j} = w_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{w_{i,j}+d_{i,j}}$, which is from Equations (1) and (4), is an increasing function of $w_{i,j}$, it holds that $w_{i,j} \geq c_{i,j}$ is equivalent to $b_{i,j} \geq c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j}+d_{i,j}}$. Further due to $d_{i,j} \geq \hat{c}_{i,j}$, we set

$$b_{i,j} \geq c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j} + \hat{c}_{i,j}}, \tag{5}$$

which implies $b_{i,j} \geq c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j}+d_{i,j}}$ and $w_{i,j} \geq c_{i,j}$.

Utility Functions. Server S aims to maximize its total wage $\sum_{i=1}^{m} \sum_{j=1}^{n_i} w_{i,j}$ under the constraints of (1), (4), (2), (3) and (5). From (1) and (4), it holds that $b_{i,j} = w_{i,j} + \frac{c_{j,j}\hat{c}_{i,j}}{w_{i,j}+d_{i,j}}$ which can be written as a quadratic equation for variable $w_{i,j}$ as $w_{i,j}^2 + w_{i,j}(d_{i,j} - b_{i,j}) + c_{j,j}\hat{c}_{i,j} - b_{i,j}d_{i,j} = 0$. Then, we have

$$w(b_{i,j}, d_{i,j}) = \frac{b_{i,j} - d_{i,j} + \sqrt{(b_{i,j} + d_{i,j})^2 - 4c_{i,j}\hat{c}_{i,j}}}{2}. \tag{6}$$

Therefore, the utility of server S is $U_s(A_s, A_{c,1}, \cdots, A_{c,m}) = \sum_{i=1}^{m} \sum_{j=1}^{n_i} w(b_{i,j}, d_{i,j})$. Each C_i aims to minimize the expected time for verifying its n_i tasks. For each task $t_{i,j}$, the expected verification time, denoted as $T_{i,j}$, is

$$T_{i,j}(b_{i,j}, d_{i,j}) = k \cdot (b_{i,j} - w(b_{i,j}, d_{i,j})) = k \cdot \frac{b_{i,j} + d_{i,j} - \sqrt{(b_{i,j} + d_{i,j})^2 - 4c_{i,j}\hat{c}_{i,j}}}{2}.$$

Then, the utility of client C_i is defined as $U_{c,i}(A_s, A_{c,i}) = \sum_{j=1}^{n_i}[T_{i,j}(b_{i,j}, d_{i,j})]$.

Nash Equilibrium of the Game. A Nash equilibrium of the game is a combination of action, denoted as $(A{*}_s,\ A{*}_{c,1},\ \cdots, A{*}_{c,m})$, taken by the server and the clients respectively, such that: for the server and any $A_s \neq A{*}_s$, $U_s(A_s, A{*}_{c,1}, \cdots, A{*}_{c,m}) \leq U_s(A{*}_s, A{*}_{c,1}, \cdots, A{*}_{c,m})$; for each client $i \in \{1, \cdots, m\}$ and any $A_{c,i} \neq A{*}_{c,i}$, $U_{c,i}(A{*}_s, A_{c,i}) \leq U_{c,i}(A{*}_s, A{*}_{c,i})$.

4 Setting I: Server S vs Single Client C_i

4.1 Client's Optimization Problem

The client's purpose is to minimize her utility, i.e., the expected time for verifying her tasks. Hence, the client's optimization problem is as follows. (Note: parameter k is ignored for the simplicity of exposition.)

$$\min \sum_{j=1}^{n_i} \frac{b_{i,j} + d_{i,j} - \sqrt{(b_{i,j} + d_{i,j})^2 - 4c_{i,j}\hat{c}_{i,j}}}{2} \tag{7}$$

$$s.t., \sum_{j=1}^{n_i} d_{i,j} = d;\ \ \sum_{j=1}^{n_i} b_{i,j} = b_i; d_{i,j} \geq \hat{c}_{i,j};\ b_{i,j} \geq c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j} + \hat{c}_{i,j}}. \tag{8}$$

4.2 Server's Optimization Problem

The server's purpose is also to maximize its utility, i.e., the total wage earned from the client. Hence, its optimization problem is as follows.

$$\max \sum_{j=1}^{n_i} \frac{b_{i,j} - d_{i,j} + \sqrt{(b_{i,j} + d_{i,j})^2 - 4c_{i,j}\hat{c}_{i,j}}}{2},\ s.t.,\ constraints\ (8). \tag{9}$$

Note that, the sum of the above two objective functions is

$$(7) + (9) = \sum_{j=1}^{n_i} b_{i,j} = b_i.$$

Hence, the objective of the server's optimization problem can be re-written to

$$\max\ b_i - \sum_{j=1}^{n_i} \frac{b_{i,j} + d_{i,j} - \sqrt{(b_{i,j} + d_{i,j})^2 - 4c_{i,j}\hat{c}_{i,j}}}{2},$$

which is further equivalent to min $\sum_{j=1}^{n_i} \frac{b_{i,j}+d_{i,j}-\sqrt{(b_{i,j}+d_{i,j})^2-4c_{i,j}\hat{c}_{i,j}}}{2}$. Therefore, the above two optimization problems are equivalent: a solution to the client's optimization problem is also a solution to the server's optimization problem; thus it is also the Nash equilibrium of the game.

4.3 Proposed Algorithm

Due to the equivalence of the above two optimization problems, we only need to solve one of them. Next, we develop the algorithm, formally presented in Algorithm 1, to find the solution to the client's optimization problem. The core of the algorithm is to solve the following optimization problem, which is rewritten from the afore-presented client's optimization problem.

$$\begin{aligned} &\min \sum_{j=1}^{n_i} f(s_{i,j}, i, j) \\ &where\ f(x,i,j) = \frac{x-\sqrt{x^2-4c_{i,j}\hat{c}_{i,j}}}{2} \\ &s.t. s_{i,j} = b_{i,j} + d_{i,j}\ and\ constraints\ (8) \end{aligned} \tag{10}$$

Note that, $f(x,i,j)$ is the client's utility associated with each task $t_{i,j}$, when the task is assigned with x as the sum of $b_{i,j}$ and $d_{i,j}$. In the algorithm, we also use a partial derivative function of $f(x,i,j)$, which is defined as

$$f'(x,i,j) = \frac{\partial f(x,i,j)}{\partial x}. \tag{11}$$

After the client and server exchange with each other their budget and deposit (i.e., b_i and d), they each run Algorithm 1 to optimally allocate $b_i + d$ to the n_i tasks, i.e., each task $t_{i,j}$ is assigned with budget $b_{i,j}$ and deposit $d_{i,j}$ where $\sum_{j=1}^{n_i} b_{i,j} = b_i$ and $\sum_{j=1}^{n_i} d_{i,j} = d$, with the goal of maximizing the client's utility. Intuitively, the algorithm runs in the following three phases:

In the first phase, each task $t_{i,j}$ is assigned an initial value for $s_{i,j}$, which denotes the sum of $b_{i,j}$ and $d_{i,j}$. Here, the initial value is set to $\hat{c}_{i,j}+c_{i,j}+\frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j}+\hat{c}_{i,j}}$ in order to satisfy constraints (8). After this phase completes, $s = b_i + d - \sum_{j=1}^{n_i}(\hat{c}_{i,j} + c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j}+\hat{c}_{i,j}})$ remains to be allocated in the second phase.

In the second phase, s is split into units each of size δ and the units are further assigned to the tasks step by step. Specifically, with each step, one remaining unit is assigned to task $t_{i,j}$ whose $f'(s_{i,j}, i, j)$ is the minimal among all the tasks; this way, the units are assigned in a greedy manner to maximize the total utility of all the n_i tasks.

After the b_i+d have been greedily assigned to all the tasks, in the third phase, $s_{i,j}$ is further split into $b_{i,j}$ and $d_{i,j}$ such that, the shorter verification time a task has, the larger deposit is assigned to it. This way, the server's deposit can be reclaimed as soon as possible from the tasks.

Algorithm 1. Optimizing Resource Allocation (Server S v.s. Client C_i with Static Task Set)

Input: b_i - total budget of client C_i; d - total deposit of server S; n_i - total number of tasks; task set $\{t_{i,1}, \cdots, t_{i,n_i}\}$ and associated costs $\{c_{i,1}, \cdots, c_{i,n_i}\}$ and $\{\hat{c}_{i,1}, \cdots, \hat{c}_{i,n_i}\}$.
Output: $\{b_{i,1}, \cdots, b_{i,n_i}\}$ and $\{d_{i,1}, \cdots, d_{i,n_i}\}$.
Phase I: Initialization.

1: **for** $j \in \{1, \cdots, n_i\}$ **do**
2: $\quad s_{i,j} \leftarrow (\hat{c}_{i,j} + c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j}+\hat{c}_{i,j}})$ ▷ meet constraints (8)

Phase II: Greedy Allocation of the Remaining Fund.

1: $s \leftarrow [b_i + d - \sum_{j=1}^{n_i}(\hat{c}_{i,j} + c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j}+\hat{c}_{i,j}})]$ ▷ remaining fund to distribute
2: **while** $s \geq \delta$ **do** ▷ distribute remaining fund in unit δ
3: $\quad j* \leftarrow \arg\min_{j \in \{1,\cdots,n_i\}} f'(s_{i,j}, i, j)$
4: $\quad s_{i,j*} \leftarrow (s_{i,j*} + \delta)$; $s \leftarrow (s - \delta)$

Phase III: Splitting Sum to Budget/Deposit.

1: $d' \leftarrow d - \sum_{j=1}^{n_i} \hat{c}_{i,j}$
2: $tempSet = \{1, \cdots, n_i\}$
3: **while** $tempSet \neq \emptyset$ **do**
4: $\quad j* = \arg\min_{j \in \{1,\cdots,n_i\}} f(s_{i,j}, i, j)$ ▷ find the task with the shortest verification time
5: $\quad x \leftarrow \min\{d', s_{i,j*} - \hat{c}_{i,j*} - (c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j}+\hat{c}_{i,j}})\}$
6: $\quad d_{i,j*} \leftarrow (\hat{c}_{i,j*} + x)$ ▷ assign as much deposit to task with the shortest verification time
7: $\quad b_{i,j*} \leftarrow (s_{i,j*} - d_{i,j*})$
8: $\quad d' \leftarrow (d' - x)$
9: $\quad tempSet \leftarrow (tempSet - \{j*\})$

4.4 Analysis

It can be proved that Algorithm 1 finds an optimal solution for the client's optimization problem (which is also a solution for the server's optimization problem), and the solution is a Nash equilibrium of the game between the client and the server. We develop the proof in the following steps. First, we introduce an optimization problem, as follows, which is relaxed from (10):

$$\min \sum_{j=1}^{n_i} f(s_{i,j}, i, j) = \sum_{j=1}^{n_i} \frac{s_{i,j} - \sqrt{s_{i,j}^2 - 4c_{i,j}\hat{c}_{i,j}}}{2} \tag{12}$$

$$s.t. - s_{i,j} + c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j} + \hat{c}_{i,j}} + \hat{c}_{i,j} \leq 0. \tag{13}$$

Second, we derive the following Lemmas.

Lemma 1. *The optimization problem defined in* (12) *has a unique solution.*

Lemma 2. *Phases I and II of Algorithm* 1 *find the unique solution to the optimization problem defined in* (12).

Lemma 3. *Phase III of Algorithm 1 converts a solution of the problem defined in (12) to a solution of the problem defined in (10).*

Based on the above lemmas, we therefore have the following theorem:

Theorem 2. *Algorithm 1 finds a solution of the problem defined in (10).*

Finally, it can also be proved the following theorem:

Theorem 3. *Algorithm 1 finds a Nash equilibrium of the game between server S and client C_i.*

The proofs for the above lemmas and theorems can be found in [27].

5 Setting II: Server S vs Clients $C_1, \cdots, C_m$

Different from the previous context of single client, optimizing for the server's utility and for each client's utility are not equivalent. So we cannot solve it in one step. Instead, we tackle the problem in two steps: we first optimize for the server's utility, which produces an allocation of the server's deposits to the clients; then, we optimize for each client's utility based on the client's budget and the deposit allocated by the server.

5.1 Algorithm

We propose an algorithm, formally presented in Algorithm 3, which runs in the following two steps.

First, we solve the server's optimization problem, which produces the optimal allocation of the server's deposits to the clients that maximizes the server's wages. Thus, the optimization problem can be defined as follows:

$$\begin{aligned} &\max \sum_{i=1}^{m}\sum_{j=1}^{n_i} w(b_{i,j}, d_{i,j}), \textit{where } w(x,y) \text{ is defined as in (6)} \\ &s.t. \text{ constraints (8).} \end{aligned} \tag{14}$$

Because

$$\begin{aligned} &\sum_{i=1}^{m} b_i - \sum_{i=1}^{m}\sum_{j=1}^{n_i} w(b_{i,j}, d_{i,j}) = \sum_{i=1}^{m}\sum_{j=1}^{n_i}[b_{i,j} - w(b_{i,j}, d_{i,j})] \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n_i} \frac{b_{i,j} + d_{i,j} - \sqrt{(b_{i,j}+d_{i,j})^2 - 4c_{i,j}\hat{c}_{i,j}}}{2} = \sum_{i=1}^{m}\sum_{j=1}^{n_i} f(b_{i,j}+d_{i,j}, i, j), \end{aligned}$$

the objective function of the above optimization problem, i.e., (14), is equivalent to min $\sum_{i=1}^{m}\sum_{j=1}^{n_i} f(b_{i,j}+d_{i,j}, i, j)$. Furthermore, let $s_{i,j} = b_{i,j} + d_{i,j}$, and then the above optimization problem can be converted to:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n_i} f(s_{i,j}, i, j), where\ f(x,i,j) \text{ is defined as in (10)},$$

$$s.t.$$

$$s_{i,j} \geq c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j}+\hat{c}_{i,j}} + \hat{c}_{i,j}, \sum_{j=1}^{n_i} s_{i,j} \geq b_i + \sum_{j=1}^{n_i} \hat{c}_{i,j}, \sum_{i=1}^{m}\sum_{j=1}^{n_i} s_{i,j} = \sum_{i=1}^{m} b_i \quad (15)$$

Here, the constraints are derived from constraints (8). This optimization problem can be solved in three phases, as formally presented in Algorithm 2.

Algorithm 2. Optimal Splitting of Deposit (Server S v.s. Client C_i, $i = 1, \cdots, m$, with Static Task Set)

Input: b_i - total budget of each client C_i; d: total deposit of server S; n_i - total number of tasks from each C_i; task set $\{t_{1,1}, \cdots, t_{1,n_1}, \cdots, t_{m,1}, \cdots, t_{m,n_m}\}$ and associated costs $\{c_{1,1}, \cdots, c_{1,n_1}, \cdots, c_{m,1}, \cdots, c_{m,n_m}\}$ and $\{\hat{c}_{1,1}, \cdots, \hat{c}_{1,n_1}, \cdots, \hat{c}_{m,1}, \cdots, \hat{c}_{m,n_m}\}$.
Output: deposit d_i allocated to each client C_i.
Phase I: Initialization.

1: **for** $i \in \{1, \cdots, m\}$ **do**
2: **for** $j \in \{1, \cdots, n_i\}$ **do**
3: $s_{i,j} \leftarrow (\hat{c}_{i,j} + c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j}+\hat{c}_{i,j}})$

Phase II: Greedy Allocation of Clients' Remaining Budgets.

1: **for** $i \in \{1, \cdots, m\}$ **do**
2: $b'_i \leftarrow b_i - \sum_{j=1}^{n_i}(c_{i,j} + \frac{c_{i,j}\hat{c}_{i,j}}{c_{i,j}+\hat{c}_{i,j}})$
3: **while** $b'_i \geq \delta$ **do**
4: $j* = \arg\min_{j \in \{1,\cdots,n_i\}} f'(s_{i,j}, i, j)$.
5: $s_{i,j*} \leftarrow (s_{i,j*} + \delta)$; $b'_i \leftarrow (b'_i - \delta)$

Phase III: Greedy Allocation of Remaining Deposit to tasks.

1: $d' \leftarrow d - \sum_{i=1}^{m}\sum_{j=1}^{n_i} \hat{c}_{i,j}$
2: $TS = \{(1,1), \cdots, (1,n_1), \cdots, (m,1), \cdots, (m,n_m)\}$
3: **while** $d' \geq \delta$ **do**
4: $(i*, j*) = \arg\min_{(i,j) \in TS} f'(s_{i,j}, i, j)$
5: $s_{i*,j*} \leftarrow (s_{i*,j*} + \delta)$; $d' \leftarrow (d' - \delta)$

Phase IV: Preparing the Output.

1: **for** $i \in \{1, \cdots, m\}$ **do**
2: $d_i \leftarrow \sum_{j=1}^{n_i} s_{i*,j*} - b_i$

In the second step, it is already known the server's deposits allocated to the clients. Because the budget of each client is also known, each server-client pair can run Algorithm 1, presented in the previous section, to find out the optimal allocation of budget/deposit to the client's tasks to minimize the client's utility.

Algorithm 3. Optimal Resource Allocation (Server S v.s. Client C_i, $i = 1, \cdots, m$, with Static Task Set)

Input: b_i - total budget of each client C_i; d - total deposit of server S; $\{n_i\}$ - total number of tasks from each C_i for $i = 1, \cdots, m$; task set $\{t_{1,1}, \cdots, t_{1,n_1}, \cdots, t_{m,1}, \cdots, t_{m,n_m}\}$ and associated costs $\{c_{1,1}, \cdots, c_{1,n_1}, \cdots, c_{m,1}, \cdots, c_{m,n_m}\}$ and $\{\hat{c}_{1,1}, \cdots, \hat{c}_{1,n_1}, \cdots, \hat{c}_{m,1}, \cdots, \hat{c}_{m,n_m}\}$.
Output: $\{b_{1,1}, \cdots, b_{1,n_1}, \cdots, b_{m,1}, \cdots, b_{m,n_m}\}$ and $\{d_{1,1}, \cdots, d_{1,n_1}, \cdots, d_{m,1}, \cdots, d_{m,n_m}\}$.

1: $\{d_1, \cdots, d_m\} \leftarrow$ Algorithm 2
2: **for** $i \in \{1, \cdots, m\}$ **do** ▷ for each client
3: $(\{b_{i,1}, \cdots, b_{i,n_i}\}, \{d_{i,1}, \cdots, d_{i,n_i}\}) \leftarrow$ Algorithm 1.

5.2 Analysis

To analyze the solution, the following can be proved: First, the optimization problem defined in (15) has only one unique solution. Second, Algorithm 2 solves the optimization problem defined in (15). Third, the optimization problem defined in (15) is equivalent to the one defined in (14). Finally, the budget and deposit allocation strategy produced by Algorithm 3 is a Nash equilibrium.

Lemma 4. *The optimization problem defined in* (15) *has one unique solution.*

Theorem 4. *Phases I-III of Algorithm 2 solves optimization problem* (15)*.*

Lemma 5. *Optimization problem* (14) *is equivalent to that defined in* (15)*.*

Theorem 5. *Algorithm 3 finds a Nash Equilibrium for the game between server S and m clients $C_1, \cdots, C_m$.*

The proofs for the above lemmas and theorems can be found in [27].

6 Conclusions and Future Works

In this paper, we study the verifiable computation outsourcing problem in the setting where a cloud server services a set of tasks submitted by multiple clients. We adopt a game-based model, where the cloud server should make a deposit for each task it takes, each client should allocate a budget that includes the wage paid to the server and the possible cost for hiring TTP for each task it submits, and every party (i.e., each of the server and the clients) has its limited fund that can be used for either deposits or task budgets. We study how the funds should be optimally allocated to achieve the three-fold goals: a rational cloud server should honestly compute each task it takes; the server's wages earned from computing the tasks are maximized; and the overall delay experienced by each task for verifying her tasks is minimized. Specifically, we apply game theory to formulate the optimization problems, and develop the optimal or heuristic solutions for two application scenarios: one client outsources a set of tasks to

the server; multiple clients outsource a set of tasks to the server. For each of the solutions, we analyze the solutions through rigorous proofs.

In the future, we will study in more depth the setting where there are multiple clients submitting dynamic sequences of tasks to the server. As it is challenging to develop optimal solution for the currently-defined general setting, we will explore to refine the problem with reasonable constraints and then develop an optimal solution for it.

Acknowledgement. The work is partly supported by NSF under grant CNS-1844591.

References

1. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_25
2. Parno, B., Raykova, M., Vaikuntanathan, V.: How to delegate and verify in public: verifiable computation from attribute-based encryption. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 422–439. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_24
3. Catalano, D., Fiore, D.: Practical homomorphic MACs for arithmetic circuits. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 336–352. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_21
4. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: IEEE Symposium on Security and Privacy, pp. 238–252. IEEE (2013)
5. Abadi, A., Terzis, S., Dong, C.: VD-PSI: verifiable delegated private set intersection on outsourced private datasets. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 149–168. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_9
6. Costello, C., et al.: Geppetto: versatile verifiable computation. In: IEEE Symposium on Security and Privacy, pp. 253–270. IEEE (2015)
7. Fiore, D., Fournet, C., Ghosh, E., Kohlweiss, M., Ohrimenko, O., Parno, B.: Hash first, argue later: adaptive verifiable computations on outsourced data. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1304–1316 (2016)
8. Setty, S.T., McPherson, R., Blumberg, A.J., Walfish, M.: Making argument systems for outsourced computation practical (sometimes). NDSS **1**(9), 17 (2012)
9. Setty, S., Vu, V., Panpalia, N., Braun, B., Blumberg, A.J., Walfish, M.: Taking proof-based verified computation a few steps closer to practicality. In: 21st USENIX Security Symposium (USENIX Security 2012), pp. 253–268 (2012)
10. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. J. ACM (JACM) **62**(4), 1–64 (2015)
11. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_2
12. Wahby, R.S., et al.: Full accounting for verifiable outsourcing. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 2071–2086 (2017)

13. Brandenburger, M., Cachin, C., Kapitza, R., Sorniotti, A.: Blockchain and trusted computing: problems, pitfalls, and a solution for hyperledger fabric (2018). arXiv preprint: arXiv:1805.08541
14. Xiao, Y., Zhang, N., Lou, W., Hou, Y.T.: Enforcing private data usage control with blockchain and attested off-chain contract execution (2019). arXiv preprint: arXiv:1904.07275
15. Cheng, R., et al.: Ekiden: a platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, pp. 185–200 (2019)
16. Tramer, F., Boneh, D.: Slalom: Fast, verifiable and private execution of neural networks in trusted hardware (2018). arXiv preprint: arXiv:1806.03287
17. Canetti, R., Riva, B., Rothblum, G.N.: Practical delegation of computation using multiple servers. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 445–454 (2011)
18. Avizheh, S., Nabi, M., Safavi-Naini, R., Venkateswarlu, M.K.: Verifiable computation using smart contracts. In: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, pp. 17–28 (2019)
19. Nix, R., Kantarcioglu, M.: Contractual agreement design for enforcing honesty in cloud outsourcing. In: Grossklags, J., Walrand, J. (eds.) GameSec 2012. LNCS, vol. 7638, pp. 296–308. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34266-0_18
20. Pham, V., Khouzani, M.H.R., Cid, C.: Optimal contracts for outsourced computation. In: Poovendran, R., Saad, W. (eds.) GameSec 2014. LNCS, vol. 8840, pp. 79–98. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12601-2_5
21. Belenkiy, M., Chase, M., Erway, C.C., Jannotti, J., Küpçü, A., Lysyanskaya, A.: Incentivizing outsourced computation. In: Proceedings of the 3rd International Workshop on Economics of Networked Systems, pp. 85–90 (2008)
22. M. Khouzani, V. Pham, and C. Cid, "Incentive engineering for outsourced computation in the face of collusion. In: Proceedings of WEIS (2014)
23. Küpçü, A.: Incentivized outsourced computation resistant to malicious contractors. IEEE Trans. Depend. Secure Comput. **14**(6), 633–649 (2015)
24. Dong, C., Wang, Y., Aldweesh, A., McCorry, P., van Moorsel, A.: Betrayal, distrust, and rationality: smart counter-collusion contracts for verifiable cloud computing. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 211–227 (2017)
25. Liu, P., Zhang, W.: A new game theoretic scheme for verifiable cloud computing. In: IEEE 37th International Performance Computing and Communications Conference (IPCCC), pp. 1–8. IEEE (2018)
26. Liu, P., Zhang, W.: Game theoretic approach for secure and efficient heavy-duty smart contracts. In: 2020 IEEE Conference on Communications and Network Security (CNS), pp. 1–9. IEEE (2020)
27. Liu, P., Ma, X., Zhang, W.: Optimizing fund allocation for game-based verifiable computation outsourcing. CoRR, vol. abs/2103.06440 (2021). https://arxiv.org/abs/2103.06440